1.socket在server端要用bind绑定地址，那么在client端可不可以用bind绑定地址来用一个固定的端口来与server通信

可以，但没必要。client用socket函数创建出套接字，然后调用connect函数连接服务器端时会自动分配一个空闲的端口。如果在connect之前调用bind给客户端套接字绑定了地址，那么要保证绑定的这个地址端口没有被其他程序占用，且这样在这台电脑上只能运行一个客户端，因为同一个端口只能给一个socket使用。

2.epoll select poll的区别?为什么用epoll不用select?

select

select原理：借助内核， select 来监听， 客户端连接、数据通信事件。

select本质上是通过设置或者检查存放fd标志位的数据结构（文件描述符监听集合，位域）来进行下一步处理。这样所带来的缺点是：

1、 单个进程可监视的fd数量被限制，即能监听端口的大小有限。（位域的大小受文件描述符上限限制，1024）

一般来说这个数目和系统内存关系很大，具体数目可以cat /proc/sys/fs/file-max察看。32位机默认是1024个。64位机默认是2048.

2、 对socket进行扫描时是线性扫描，即采用轮询的方法，效率较低：

当套接字比较多的时候，每次select()都要通过遍历FD_SETSIZE个Socket来完成调度,不管哪个Socket是活跃的,都遍历一遍。这会浪费很多CPU时间。如果能给套接字注册某个回调函数，当他们活跃时，自动完成相关操作，那就避免了轮询，这正是epoll与kqueue做的

3、需要维护一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大（文件描述符监听集合）（每次调用select，都需要把fd集合从用户态拷贝到内核态，这个开销在fd很多时会很大)

优点： **跨平台。**win、linux、macOS、Unix、类Unix、mips 就是因为可以跨平台select才一直没有被淘汰

poll

poll本质上和select没有区别

它没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有select的其他缺点：

1、对socket进行扫描时是线性扫描，即采用轮询的方法，效率较低

2、大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义。

poll还有一个特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd。

epoll

epoll存套接字fd的数据结构是epoll对象，这个对象实际上是一个二叉树，这样也突破了1024文件描述符的限制。

epoll有EPOLLLT和EPOLLET两种触发模式，LT是默认的模式，ET是“高速”模式。LT模式下，只要这个fd还有数据可读，每次epoll_wait都会返回它的事件，提醒用户程序去操作，而在ET（边缘触发）模式中，它只会提示一次，直到下次再有数据流入之前都不会再提示了，无 论fd中是否还有数据可读。所以在ET模式下，read一个fd的时候一定要把它的buffer读光，也就是说一直读到read的返回值小于请求值，或者 遇到EAGAIN错误。还有一个特点是，epoll使用“事件”的就绪通知方式，通过epoll_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll_wait便可以收到通知。

epoll为什么要有EPOLLET触发模式?

如果采用EPOLLLT模式的话，系统中一旦有大量你不需要读写的就绪文件描述符，它们每次调用epoll_wait都会返回，这样会大大降低处理程序检索自己关心的就绪文件描述符的效率.。而采用EPOLLET这种边沿触发模式的话，当被监控的文件描述符上有可读写事件发生时，epoll_wait()会通知处理程序去读写。如果这次没有把数据全部读写完(如读写缓冲区太小)，那么下次调用epoll_wait()时，它不会通知你，也就是它只会通知你一次，直到该文件描述符上出现第二次可读写事件才会通知你！！！这种模式比水平触发效率高，系统不会充斥大量你不关心的就绪文件描述符

epoll的优点：

1、没有最大并发连接的限制，能打开的FD的上限远大于1024（1G的内存上能监听约10万个端口）；
2、效率提升，不是轮询的方式，不会随着FD数目的增加效率下降。只有活跃可用的FD才会调用callback函数；即Epoll最大的优点就在于它只管你“活跃”的连接，而跟连接总数无关，因此在实际的网络环境中，Epoll的效率就会远远高于select和poll。

3、 内存拷贝，利用mmap()文件映射内存加速与内核空间的消息传递；即epoll使用mmap减少复制开销。